Proceduralna generacija 3D okruženja, s prikazom u formi piksel umjetnosti – PixFER OpenWorlds

Tehnička dokumentacija

Verzija 1.0

Studentski tim:

Marin Banožić

Maksimilijan Hižman

Dorijan Jančić

Marko Lujo

Tin Salopek

Vito Vrbić

Nastavnik: Prof. dr. Sc. Željka Mihajlović

[1. Opis razvijenog proizvoda 3](#_Toc188194494)

[1.1 PixFER alat 3](#_Toc188194495)

[1.2 OpenWorlds alat za Unity 5](#_Toc188194496)

[1.3 Demonstracijski Unity projekt 7](#_Toc188194497)

[2. Tehničke značajke 8](#_Toc188194498)

[2.1 Tehničke značajke PixFER alata za Unity 8](#_Toc188194499)

[2.1.1 PRRenderPipeline.cs 8](#_Toc188194500)

[2.1.2 Renderer.cs 8](#_Toc188194501)

[2.1.3 RenderPass.cs 8](#_Toc188194502)

[2.1.4 Lights.cs 8](#_Toc188194503)

[2.1.5 Kratak opis korištenih prolaza i sjenčara 8](#_Toc188194504)

[2.2 Tehničke značajke OpenWorlds alata za Unity 10](#_Toc188194505)

[2.2.1 OW\_ChunkLoader.cs, OW\_Chunk.cs, OW\_ChunkState.cs 10](#_Toc188194506)

[2.2.2 OW\_ChunkEditor.cs, OW\_Sculptor.cs, OW\_Painter.cs, OW\_Decorator.cs 12](#_Toc188194507)

[2.2.3 OW\_PathPainter.cs 13](#_Toc188194508)

[3. Upute za korištenje 15](#_Toc188194509)

[3.1 Korištenje PixFER alata u drugim Unity projektima 15](#_Toc188194510)

[3.2 Korištenje OpenWorlds alata u drugim Unity projektima 15](#_Toc188194511)

[4. Literatura 18](#_Toc188194512)

[5. Slike 18](#_Toc188194513)

Tehnička dokumentacija

# Opis razvijenog proizvoda

Alati PixFER i OpenWorlds primarno su razvijeni kako bi se mogli koristiti odvojeno (kompletno neovisno jedan o drugome), ali je njihova međusobna interakcija uzeta u obzir kako bi njihovo zajedničko korištenje moglo omogućiti stvaranje beskonačnih interaktivnih 3D okruženja u kojima korisnik (ili igrač) može istraživati beskonačan svijet u formi piksel umjetnosti s parametrima stvaranja svijeta po želji.

## PixFER alat

PixFER alat razvijen je za razvojno okruženje Unity i omogućava iscrtavanje 3D okruženja na način da se to iscrtavanje pridržava formi piksel umjetnosti; alat omogućava da obična, ne-stilizirana 3D okolina igraču izgleda kao ručno nacrtana slika u formi piksel umjetnosti.

Korištenjem prilagođenog SRP-a (Scriptable Render Pipeline-a), alat omogućuje fleksibilnost u iscrtavanju, a, uz specijalizirane sjenčare, okolina je dodatno vizualno stilizirana da odgovara formi piksel umjetnosti.

Za dodatnu optimizaciju vizualnog iskustva igrača, alat dolazi s posebno konfiguriranim objektom kamere koji eliminira problem treperenja piksela prilikom kretanja kamere kroz okolinu.

Glavni dio implementacije prilagođenog SRP-a nalazi se u imenskom prostoru PR.Rendering,

koji sadrži glavne klase:

* **PRRenderPipeline**, klasa koja je zadužena za upravljanje kamerama, svjetlima te postavkama prikazivanja i pikselizacije;
* **PRRenderPipelineAsset**, koja sadrži postavke prikazivanja dostupne korisniku; te su postavke količina pikselizacije, kretanje kamera i kvaliteta sjena;
* **Renderer**, klasa koja izvršava prikazivanje jedne kamere te pokreće sve potrebne ‘render’ prolaze za prikazivanje slike;
* **Lights**, klasa koja upravlja prilagođenim načinom prikazivanja svjetala i osvjetljenjem objekata.

**Okviran opis funkcioniranja SRP-a:**

Prije pokretanja aplikacije, korisnik postavlja željene parametre objekta RenderPipelineAsset (instanca klase PRRenderPipelineAsset) te određuje glavnu kameru scene. Pri pokretanju scene stvara se novi objekt klase PRRenderPipeline koji također stvara novi objekt klase Renderer.

Za svaku prikazanu sliku (frame), objekt PRRenderPipeline poziva svoju funkciju RenderSingleCamera() za svaku kameru u sceni. Funkcija RenderSingleCamera() prvo inicijalizira podatke o prikazu kamere za koju je pozvana, stvara opisnike teksture niske rezolucije, te sprema te informacije (i postavke PRRenderPipelineAsset-a) u strukturu RenderingData, koja je zatim proslijeđena prije-inicijaliziranom Renderer objektu.

Objekt Renderer prvo pozove svoju funkciju Setup() koja postavi sve potrebne prolaze u red, te zatim poziva funkciju Execute(), koja stvori render teksture, postavi inpute za sjenčare, i izvrši sve prolaze redoslijedom zadanim u datoteci RenderPass.cs.

Nakon izvršavanja svih blokova, objekt Renderer šalje upute naredbenom međuspremniku (command buffer), te PRRenderPipeline pošalje rezultate traženoj teksturi (ili, u slučaju glavne kamere, ekranu).

**Prolazi i sjenčari:**

Potrebne funkcije koje prolaz (pass) mora sadržavati definirane su u apstraktnoj klasi RenderPass. Prolazi služe za definiciju ulaznih i izlaznih varijabli te slanje naredbi naredbenom međuspremniku, najčešće za crtanje grafičkih elemenata scene.

Svaki prolaz također sadrži prioritet koji određuje kojim se redoslijedom izvodi u redu. Prioriteti su grupirani u faze, koje su redom izvođenja (od prve do zadnje):

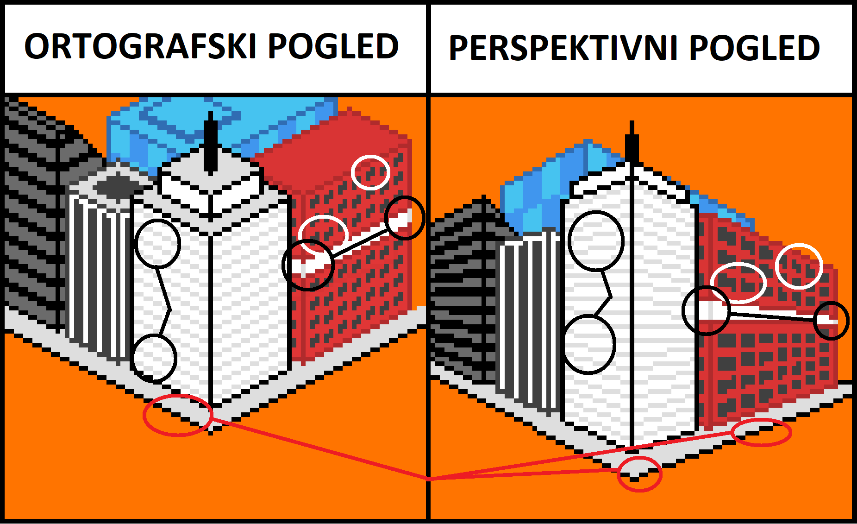
* RenderProbe;
* RenderShadows (za računanje pozicija sjena);
* RenderCameraPrePasses;
* RenderOpaque (prikazivanje neprozirnih objekata);
* RenderSkybox;
* RenderTransparent (prikazivanje prozirnih objekata);
* RenderPostProcessing (naknadni efekti zaslona);
* AfterRendering.

Budući da je prioritet tipa broj, omogućena je funkcionalnost dodatnog sortiranja prolaza unutar faze u kojoj se nalaze.

Uz sve to implementirani su i programi grafičkog procesa (sjenčari) za prikazivanje jednostavnih neosvijetljenih materijala, prilagođen sustav računanja osvjetljenja i sjena za prikaz osvijetljenih objekata, ali i dodatni efekti kao jednostavne podvodne refrakcije i pikselizirani obrub (line-art) oko objekata u Unity sceni.

**Tehnike pikselizacije:**

Za postizanje izgleda nalik piksel umjetnosti, kamere su postavljene tako da prvo prikažu svoje podatke na teksturu male rezolucije, koja je zatim uvećana za postavljenu vrijednost (ovisno o rezoluciji prozora/ekrana) te tek onda kopirana na zaslon. Kamere su također postavljene na ortografski prikaz kako bi prikazani objekti bili konzistentni (u piksel umjetnosti perspektivni prikaz nije česta pojava te zbog toga perspektivan prikaz u piksel umjetnosti izgleda „čudno” kad je i ručno nacrtan, pa smo ga odlučili izbjeći u potpunosti). Na slici 1.1 vidi se primjer razlike ortografskog i perspektivnog pogleda u ručno nacrtanoj piksel umjetnosti.



Slika 1.1: Usporedba ortografskog i perspektivnog pogleda

u piksel umjetnosti. [[1]](#_Slike)

U crnim krugovima vidi se razlika u konzistentnosti linija (veličina i smjer), u crvenim problem podizanja piksela jednolikog pravca (na lijevoj strani slike 1.1 pravac je stalnog podizanja svaka 2 piksela, dok je na desnoj nasumično mijenjanje podizanja pravca između 1 i 3 piksela), a u bijelim vidi se problem u konzistentnosti površina gdje površine prozora na zgradi u desnoj slici imaju „negeometrijski“ oblik, a na lijevoj slici je oblik identičan svim površinama prozora.

Jedan problem s kojim smo se susreli prilikom implementacije alata bio je efekt treperenja piksela (pixel shimmering). Shvatili smo da on nastaje zbog toga što pri pomicanju kamere prikazani objekti neće zauzimati isto mjesto na mreži piksela nisko-rezolucijske teksture te će biti nekonzistentno prikazani na ekranu.

Zbog toga je u klasi PRRenderPipeline implementirana metoda PixelCalculations() koja premješta matricu pogleda (view) kamere na cijele piksele, i također pomiče sliku na zaslonu ovisno o poziciji kamere kako bi se njezino pomicanje činilo glađim.

Funkcija se može isključiti u PRRenderPipelineAsset u slučaju da je efekt svjetlucanja piksela poželjan, te se također ručno može postaviti veličina jednog piksela nisko-rezolucijske teksture na zaslonu.

## OpenWorlds alat za Unity

OpenWorlds alat razvijen je za razvojno okruženje Unity i pruža funkcionalnost generiranja beskrajno-velikih otvorenih svjetova iz početnih parametara putem proceduralnog generiranja okoline. Ovaj alat korisniku daje mogućnost korištenja već definiranih parametara ili stvaranja kompletno novih parametara koji utječu na stvaranje svijeta prema točno određenim željama korisnika.

U jezgri alata nalaze se:

* Unity skripta koja omogućava dinamičko učitavanje i brisanje segmenata svijeta za optimalno ponašanje Unity projekta (OW\_ChunkLoader.cs);
* Unity skripta koja učitava početne parametre i pomoću njih uređuje segmente svijeta po potrebi (OW\_ChunkEditor.cs);
* Apstraktne skripte za uređivanje visine segmenata (OW\_Sculptor.cs), postavljanje tekstura na segmente (OW\_Painter.cs), stavljanje dodatnih objekata (kao što su drveće i kamenje) na segmente (OW\_Decorator.cs);
* pomoćna klasa koja obuhvaća najvažnije funkcionalnosti samog segmenta svijeta (OW\_Chunk.cs)

Kao dodatak alatu, dokaz njegovog funkcioniranja, ali i olakšavanje korištenja alata korisnicima koji nemaju vještine u skriptiranju, ubačene su i dodatne skripte koje se mogu koristiti za stvaranje svjetova, a implementiraju funkcionalnosti apstraktnih skripti OW\_Sculptor.cs, OW\_Painter.cs, te OW\_Decorator.cs:

**OW\_PerlinNoiseSculptor.cs:**

* Skripta koja koristi Perlinov šum za stvaranje prirodnih visinskih varijacija na segmentima svijeta.
* Od parametara za stvaranje svijeta sadrži:
  + Skala (Scale) - kontrolira razinu detalja šuma; veće vrijednosti rezultiraju glađim i širim varijacijama visina.
  + Visinska Skala (HeightMultiplier) - određuje maksimalnu visinu terena; veće vrijednosti rezultiraju izraženijim planinama.
  + Oktave (Octaves) - broj slojeva Perlinovog šima koji se koriste za dodavanje detalja; više slojeva povećavaju složenost terena zbog većeg broja detalja.
  + Perzistencija (Persistence) - utječe na količinu doprinosa svakog idućeg sloja ukupnome šumu/rezultatu, a zauzima vrijednosti između 0 i 1.
  + Lakunarnost (Lacunarity) - kontrolira razliku u frekvenciji između oktava; veće vrijednosti rezultiraju oštrijim prijelazima i većom količinom detalja.
  + Offset - pomiče početnu točku generiranja šuma; generalno se ne koristi u alatu, ali je korištenje implementirano za korisničke potrebe.

**OW\_QuickErosionSculptor.cs :**

* Skripta koja omogućava dodavanje realističnih visinskih promjena kao što su poravnavanje dolina, izoštravanje vrhova, simulacije erozije i dodavanje detalja grebena na već generirane visinske varijacije.
* Od parametara za stvaranje svijeta sadrži:
  + Visinska Skala (HeightMultiplier) – Skala koja kontrolira visinsku prilagodbu terena; veće vrijednosti rezultiraju višim terenom.
  + Snaga Erozije (ErosionStrength) – Intenzitet erozije; veće vrijednosti uzrokuju jaču eroziju, stvarajući dublje i oštrije razlike u visinama terena.
  + Detalji Grebena (RidgeDetail) – Dodaje finije detalje na grebenima; veće vrijednosti povećavaju razlučivost detalja na vrhovima planina.
  + Frekvencija Grebena (RidgeFrequency) – Frekvencija detalja na grebenima; veće vrijednosti stvaraju češće detaljima.
  + Koraci Terasa (TerraceSteps) – Broj terasa (stepenica) koje se stvaraju na većim visinama; veće vrijednosti rezultiraju oštrijim prijelazima između terasastih visina.
  + Snaga Izravnavanja (FlatnessStrength) – Jačina izravnavanja za ravne površine; veće vrijednosti omogućuju dodatno izravnavanje područja s nižim visinama.
  + Prag Glatkog Prijelaza (BoundarySmoothingThreshold) – Prag za prijelaz između planina i ravnica; područja ispod ovog praga bit će podložna glatkoj prijelaznoj površini.
  + Faktor Glatkog Prijelaza (BoundarySmoothingFactor) – Faktor glatkoće koji kontrolira količinu izravnavanja u prijelazima između planinskih i ravnih područja.
  + Veličina Serije (BatchSize) – Broj vrhova obrađenih po jednom okviru; manja vrijednost povećava glatkoću generiranja, dok veća omogućuje bržu obradu većih svjetova.

**OW\_RiverSculptor.cs:**

* Skripta koja omogućava stvaranje rijeka na temelju grebenskog Perlinovog šuma. Skripta ovisi o PerlinNoiseSculptor skripti za generiranje "grebena" na prvoj oktavi, koji se koriste za definiranje oblika rijeka.
* Od parametara za stvaranje svijeta sadrži:
  + Visina Perlinovog Šuma (riverPerlinHeight) – Određuje vrijednost Perlinovog šuma ispod koje se generiraju rijeke. Veća vrijednost rezultira širim rijekama (zadano 0.065f).
  + Širina Rijeka(riverWidth) – Maksimalna širina rijeka u jedinicama (zadano 10). Potrebno smanjiti ako rijeke "izlaze" izvan željenog oblika.
  + Dubina Rijeka(riverDepth) – Maksimalna dubina rijeka (kao postotak maksimalne visine terena) (zadano 0.05f).
  + Dodatna Dubina Rijeka(riverExtraDepth) – Dodatna dubina za izraženije rubove rijeka (kao postotak maksimalne visine terena) (zadano 0.01f).
  + Veličina serije(BatchSize)– Broj vrhova koji se obrađuju po prikazu za bolje performanse (zadano 1000).

**OW\_HeightSlopePainter.cs:**

* Skripta koja omogućava bojanje teksture na segmente ovisno o visini i nagibu.
* Od parametara za stvaranje svijeta sadrži:
  + Razred TextureData – sadrži parametre: ime teksture (name), referencu na Texture2D objekt (texture), maksimalna visina na kojoj tekstura može biti primijenjena (heightThreshold), maksimalan nagib na kojem tekstura može biti primijenjena (slopeThreshold), polje objekata tipa Color (cachedPixels);
  + Postoje i dvije privatne metode: CachePixels – inicijalizira cachedPixels u ovisnosti o predanoj teksturi, GetCachedPixels – omogućuje dohvat pixela na temelju uv vrijednosti
  + Lista TextureData objekata (textureDataList) – određuje koje teksture mogu biti stavljene na mesh
  + Faktor skaliranja teksture (textureTiling) – određuje koliko često će se ponavljati teskstura, veći faktor povećati će ponavljanje

**OW\_PathPainter.cs:**

* Skripta koja boja puteve na segmente.
* Od parametara za stvaranje svijeta sadrži:
  + POICount - predstavlja broj POI-a koji se generira po „chunku“
  + minimalDistance - predstavlja minimalnu udaljenost koja mora biti između 2 POI-a
  + globalMaxHeight - predstavlja maksimalnu visinu koja je pronađena tijekom stvaranja POI-a na različitim chunkovima
  + minimalHeightThreshold - predstavlja uvjetnu vrijednost za stvaranje POI-a vezanu uz minimalnu visinu na kojoj se više ne smije stvarati POI
  + HeightAdjustmentRate - predstavlja mjeru do koje je dozovljeno promijeniti minimalHeightThreshold
  + maxHeightBuffer - predstavlja parametar u određivanju minimalHeightThresholda, služi za prevenciju uzimanja totalnog maksimuma visine koji se pronađe tijekom stavljanja POI-a
  + pathRadius - predstavlja radijus bojanja oko točaka na gotovim putevima
  + pathRadiusVariance - predstavlja dozvoljeno odstupanje od radijusa bojanja oko točaka na gotovim putevima
  + startColor - predstavlja početnu boju gradijentalnog bojanja
  + endColor - predstavlja završnu boju gradijentalnog bojanja

**OW\_HeightSlopeTextureDecorator.cs:**

* Skripta koja omogućava ukrašavanje terena dodavanjem objekata (dekoracija) na temelju visine, nagiba i teksture.
* Od parametara za stvaranje svijeta sadrži:
  + Filter Dekoracije(DecorationFilter) – razred koji sadrži parametre za definiranje ukrasa:
    - Prefabs – polje referenci na GameObject prefabe koji se koriste za ukrašavanje.
    - Texture – referenca na Texture2D objekt koji se koristi za uzorkovanje područja za dekoracije.
    - TextureScale – skaliranje teksture u svjetskom prostoru, određuje frekvenciju uzorkovanja (zadano 10f).
    - MinHeight i MaxHeight – minimalna i maksimalna visina na kojoj dekoracije mogu biti postavljene.
    - MinSlope i MaxSlope – minimalni i maksimalni nagib terena za postavljanje dekoracija (u stupnjevima).
    - SlopeStrength – jačina poravnanja dekoracija s nagibom terena.
    - MaxDecorations – maksimalan broj dekoracija koji se može postaviti za pojedini filter.
    - Filters – niz objekata klase DecorationFilter koji definiraju različite skupove pravila za ukrašavanje.
    - GridUnit – razmak između točaka uzorkovanja na mreži (zadano 10f).
    - OffsetRange – maksimalni slučajni pomak u uzorkovanju mreže (zadano 2f).

## Demonstracijski Unity projekt

Demonstracijski Unity projekt ima implementirane alate OpenWorlds i PixFER te u sebi sadrži kameru s kontrolama i dvije scene koje prikazuju mogućnosti koje OpenWorlds i PixFER pružaju korisniku.

# Tehničke značajke

Najvažnije tehničke značajke projekta; one za koje smo smatrali da je važno naglasiti i objasniti njihov način funkcioniranja, ovdje su navedene i objašnjene za oba alata.

## Tehničke značajke PixFER alata za Unity

### PRRenderPipeline.cs

Klasa PRRenderPipeline implementira apstraktnu metodu klase RenderPipeline:

Render(ScriptableRenderContext context, List<Camera> cameras)

Ova metoda izvršava metodu RenderSingleCamera() za svaku aktivnu kameru u sceni, te se poziva za svaki prikaz (frame).

Prije izvođenja metoda prikazivanja, metoda RenderSingleCamera() prvo isključuje geometriju scene koja je nevidljiva kameri iz kompletne geometrije za prikaz – podaci o vidljivosti objekata dohvate se pomoću metode kamere TryGetCullingParameters().

Nakon toga izvršava se privatna metoda InitializeRenderingData() koja dohvati sve podatke potrebne za prikaz, a zatim ih smjesti u odgovarajuće strukture podataka.

Sve pod-strukture spremaju se u RenderingData strukturu, te se ona prosljeđuje klasi Renderer u metodama RendererSetup() i RendererExecute().

Kroz cijeli pipeline također se prosljeđuje Unity klasa ScriptableRenderContext, koja sadrži bitne informacije potrebne za prikazivanje.

### Renderer.cs

Nakon što metoda Renderer.Setup() postavi sve prolaze (pass) u red izvršavanja, metoda Execute() postavi sve globalne matrice (model, view i projection matrice) naredbenog međuspremnika na vrijednosti koje dohvati iz strukture RenderingData.cameraData. Također su postavljene globalne varijable koje će se koristiti u sjenčarima – trenutno vrijeme i veličina teksture niske rezolucije. Nakon toga definiranim redom izvrši se red prolaza pozivanjem metode ExecuteBlock().

Klasa Renderer parcijalna je klasa te se u datoteci Render.Queue.cs nalaze dijelovi te parcijalne klase Renderer koji organiziraju i sortiraju klase RenderPass kako bi se izvršile točnim redoslijedom.

### RenderPass.cs

RenderPass apstraktna je klasa koja definira potrebnu metodu Execute() koju prolaz mora izvršiti. Execute() predaje kontekst i RenderingData prolazu koji ih u najčešćim slučajevima koristi kako bi stvorio i poslao naredbe grafičkom procesu preko naredbenog međuspremnika.

### Lights.cs

Klasa Lights sadrži implementaciju prilagođenog sustava osvjetljenja. Prije izvršavanja, klasa Renderer poziva metodu Lights.Setup() koja u naredbenom međuspremniku postavlja vrijednosti pozicije, smjera i boje glavnog svjetla, kao i sporednih svjetala u sceni. Implementirane vrste svjetala su usmjereno (ne sadrži poziciju), točkasto (ne sadrži smjer) i usmjereno točkasto svjetlo.

Prolazi i sjenčari kasnije mogu koristiti ove varijable kako bi izračunali boju prikazanih površina.

### Kratak opis korištenih prolaza i sjenčara

**Općeniti sjenčari korišteni na više mjesta (ShaderLibrary):**

* *Common.hlsl* - služi samo za dobivanje view vektora u globalnom prostoru
* *Input.hlsl* - sadrži deklaracije često korištenih ulaznih varijabli kao matrice za transformaciju vektora u pogledni prostor, podatke o zaslonu, kamere itd.
* *Lighting.hlsl* - sadrži strukturu Light koja je korištena u računanju svjetala, kao i funkcije za računanje jačine izražaja svjetla u odnosu na udaljenost i smjer, i funkcije za dobivanja strukture svjetla iz scene
* *Shading.hlsl* - računa osvjetljenje površine koristeći funkcije iz gornje datoteke
* *Shadows.hlsl* - računa dijelove scene koji nisu osvijetljeni zbog prepreka drugih objekata (usmjerene sjene, zapisane u teksturu sjena kao perspektiva svjetla koje se koristi) i zbog oblaka (generiranih sjena)

**Prolazi:**

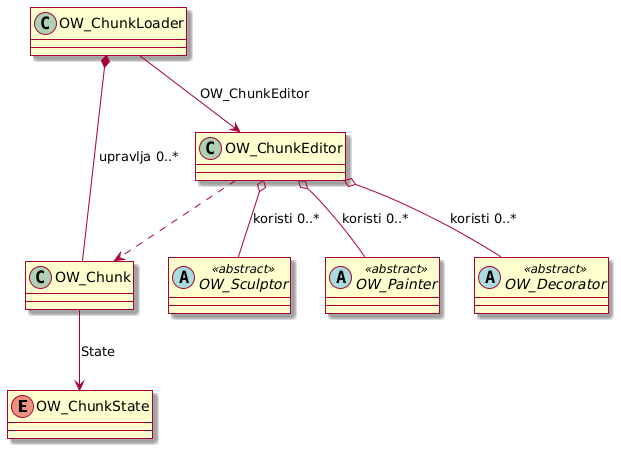
* *CopyAttachmentPass.cs* - služi za kopiranje tekstura dubine i boje te njihovog prikazivanja na kameru preko crtanja pravokutnika koji je sačinjen od kopirane teksture
* *DepthPrePass.cs* - služi za dobivanje teksture dubine (informacije koliko je određeni piksel udaljen od kamere) iz perspektive dane kamere
* D*irectionalShadowPass.cs* - Crta sjene na teksturu sjena i postavlja relevantne podatke sjenčarima
* *FinalBlitPass.cs* - Izvršava sjenčar blit koji kopira teksture boje(u koje su zapisivali prošli sjenčari i prolazi) na teksturu u koju je kamera postavljena (ili na zaslon ako je to glavna kamera)
* *ObjectPass.cs* - izvršava crtanje objekata u sceni (sjenčari se izvršavaju ovisno o materijalu koji je dodijeljen objektu)
* *ScreenSpaceOutlinesPass.cs* - Crta obrube objekata na posebnu teksturu koja se kasnije poveže s glavnom teksturom boja
* *SkyboxPass.cs* - Crta skybox, tj. teksturu koja je prikazana kada niti jedan objekt nije vidljiv kameri na pikselu
* *VolumetricLightPass.cs* - Neimplementiran prolaz koji izvršava sjenčar za računanje volumnog svjetla

**Sjenčari:**

* *Unlit.shader* - služi samo kao omot za UnlitPass
* *UnlitPass.hlsl* - jednostavan sjenčar koji premješta pozicije točaka (vertex-a) iz globalne pozicije u kamerin prostor i boji ih zadanom glavnom bojom
* *Lit.shader* - koristi LitForwardPass, DepthPass i ShadowCasterPass kako bi obojio objekte ovisno o osvjetljenju
* *LitForwardPass.hlsl* - računa boju objekata ovisno o svjetlu. Poziva funkciju shade\_opaque iz datoteke Shading.hlsl
* *DepthPass.hlsl* - Vraća dubinu (udaljenost objekta od kamere)
* *ShadowCasterPass.hlsl* - premješta točke na pozicije zaslona i mijenja njihovu dubinu
* *Water.shader* - računa sve vezano s vodom - boju površine, njezino pomicanje (određeno nasumičnim šumom), te simuliranu refrakciju svjetla nad objektima koji se nalaze ispod površine
* *Utilities/Blit.shader* - Spaja dobivene teksture boje i dubine te ih prikazuje na izlaznu teksturu. Nakon dobivanja izlazne slike, sjenčar također primjenjuje gradaciju boja tehnikom pretražne teksture - slike koja sadrži sve dostupne boje s koordinatama raspoređenim ovisno o vrijednostima komponente originalne boje. Primjenom teksture s ograničenim izborom boja prikazana slika također će sadržavati samo taj ograničen izbor.
* *PostProcessing/ScreenSpaceOutlines.shader* - Pomoću teksture dubine i boje boji piksel pored ruba objekta konstantom bojom obruba. Dobivena tekstura se zatim spaja s originalnom slikom.
* *PostProcessing/VolumetricLight.shader* - Neimplementiran sjenčar koji računa raspršivanje svjetla ovisno o atmosferi i volumenu

## Tehničke značajke OpenWorlds alata za Unity

Na slici 2.1 prikazan je klasni dijagram baze alata OpenWorlds, dok su u narednim pod poglavljima detaljno objašnjene funkcionalnosti klasa u dijagramu.



Slika 2.1: Klasni dijagram alata OpenWorlds

(bez unutarnjih atributa i metoda).[[2]](#_Slike)

### OW\_ChunkLoader.cs, OW\_Chunk.cs, OW\_ChunkState.cs

Glavna funkcionalnost OpenWorlds alata dinamičko je učitavanje i brisanje objekata tipa OW\_Chunk koje će se nadalje nazivati segmentima svijeta (ili samo segmentima). Obaveza ove funkcionalnosti pridijeljena je klasi OW\_ChunkLoader.

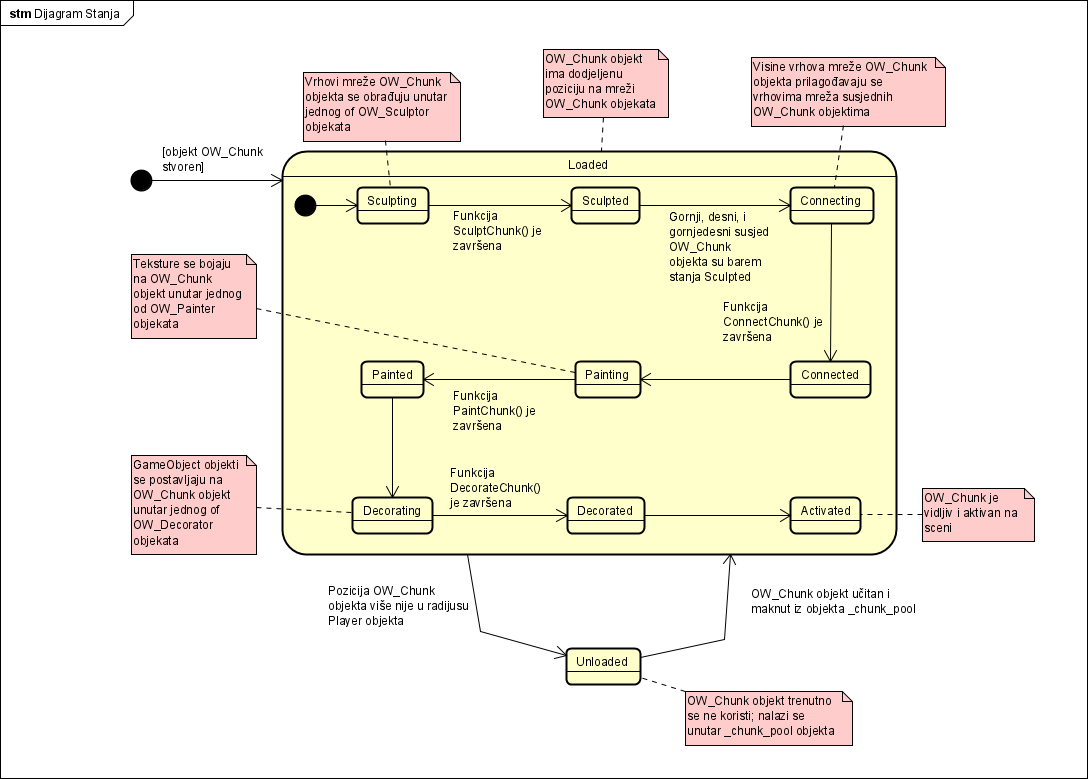
Početak rada ove skripte započinje instanciranjem dvije glavne komponente za upravljanje segmentima svijeta: red \_chunk\_pool koji u sebi može sadržavati segmente svijeta, te mapa \_loaded\_chunks čiji je ključ par koordinata, a čija je vrijednost segment svijeta.

Odmah u idućem prikazu (frame-u), skripta kreće s upravljanjem segmentima ovisno o tome gdje se nalazi objekt kamere, koliko segmenata je potrebno prikazati u okrugu kamere, te o samom stanju komponenata \_chunk\_pool te \_loaded\_chunks.

Kako bi se izbjeglo smanjivanje performansi zbog stalnog zvanja vremenski-zahtjevne funkcije Instantiate() u Unity okruženju, segmenti se “recikliraju” te se nikada ne koristi veći broj segmenata od onoliko koliko ih je potrebno za radijus u okolini kamere. Ako je objekt \_chunk\_pool prazan, a potrebno je učitati segment, onda se koristi funkcija Instantiate(), ali ako se u objektu \_chunk\_pool nalazi barem jedan segment (\_chunk\_pool.Count > 0), jedan se segment vadi iz reda, dodjeljuje mu se nova pozicija, a podatci o generiranom okruženju (3D mreža) se brišu kako bi se mogli stvoriti i generirati novi.

U Update() metodi, prvo se uvijek poziva UnloadChunks() metoda koja sve segmente koji nisu u radijusu kamere stavlja u objekt \_chunk\_pool, a tek se onda poziva metoda LoadChunks() koja učitava segmente koji još nisu učitani. Pošto je u radijusu oko igrača uvijek isti maksimum segmenata osigurano je da je broj segmenata u sceni uvijek isti ako je radijus generiranja oko kamere konstantan.

Na slici 2.2 prikazan je dijagram stanja segmenta (objekta OW\_Chunk), čija su prebacivanja iz jednog stanja u drugo zaduženja OW\_ChunkLoader skripte.



Slika 2.2: Dijagram stanja objekta OW\_Chunk.[[3]](#_Slike)

Sekundarna, ali svejedno važna funkcionalnost OW\_ChunkLoader klase povezivanje je segmenata koji su u stanju Sculpted, te čiji su susjedi (gornji, desni, gornji desni) segmenti u stanju između (uključivo) stanja Sculpted i stanja Activated. Povezivanje se radi na način da se poziva funkcija

OW\_Chunk.ConnectToNeighbors(OW\_Chunk top, OW\_Chunk topright, OW\_Chunk right)

nad segmentom kojeg se želi povezati.

Funkcija ConnectToNeighbors() koristi jednostavan algoritam povezivanja koji osigurava da će svi generirani segmenti biti prikladno spojeni. Taj algoritam prikazan je u narednom pseudokodu:

za svaki(vrh iz mreža.gornji\_vrhovi)

{

vrh = gornji\_susjed.mreža.prikladni\_donji\_vrh

}

Za svaki(vrh iz mreža.desni\_vrhovi)

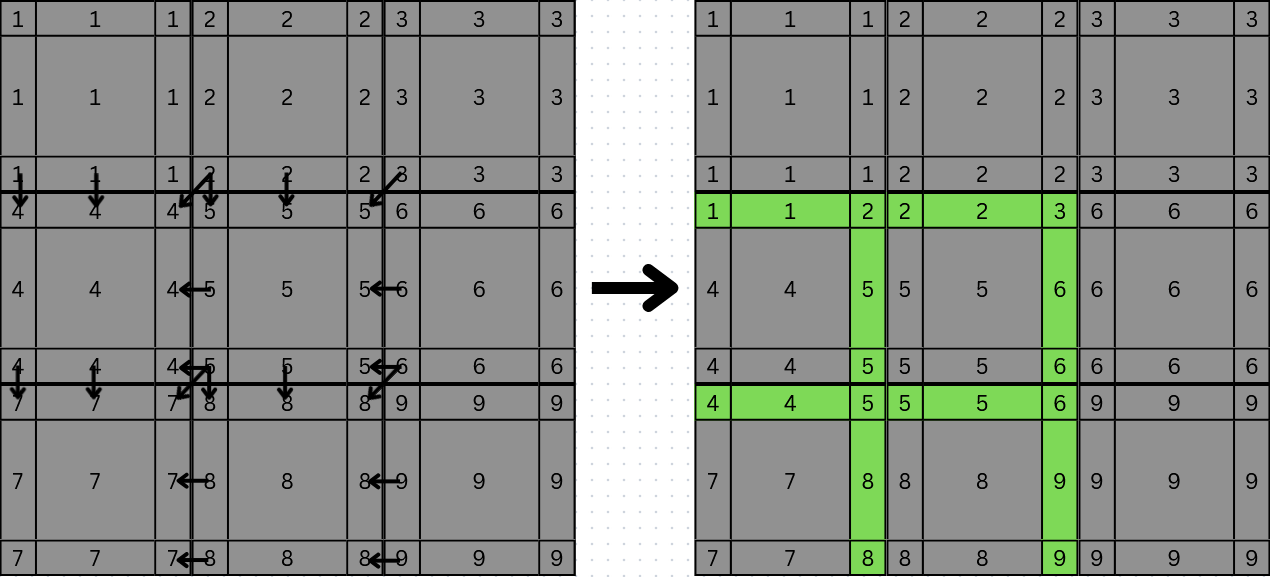
{

vrh = desni\_susjed.mreža.prikladni\_lijevi\_vrh

}

mreža.gornji\_desni\_vrh = gornji\_desni\_susjed.mreža.donji\_lijevi\_vrh

Na slici 2.3 vidljivo je kako algoritam radi, uzevši u obzir da su algoritmu podloženi samo oni segmenti koji imaju postojeće gornje, desne, i gornje-desne susjede (na prikazu to su segmenti 4, 5, 7, 8).



Slika 2.3: Vizualni prikaz rada algoritma spajanja segmenata[[4]](#_Slike)

Kako su u sceni aktivni (na ekranu vidljivi) samo oni segmenti koji su u stanju Active, a za doći u stanje Active potrebno je proći kroz algoritam spajanja, osigurano je da igrač nikada neće vidjeti segmente, a samim time niti vrhove koji nisu povezani sa susjednim segmentima.

### OW\_ChunkEditor.cs, OW\_Sculptor.cs, OW\_Painter.cs, OW\_Decorator.cs

Klasa OW\_ChunkEditor pruža javne funkcije za uređivanje visine, bojanje i dekoriranje segmenata:

public IEnumerator SculptChunk(OW\_Chunk chunk);

public IEnumerator PaintChunk(OW\_Chunk chunk);

public IEnumerator DecorateChunk(OW\_Chunk chunk);

Te funkcije koristi klasa OW\_ChunkLoader, ali kako bi ih OW\_ChunkLoader objekt mogao koristiti, pri inicijalizaciji objekta OW\_ChunkEditor, u privatna polja \_sculptors, \_painters, \_decorators sakuplja sve funkcije na istom Unity objektu, te ih unutar navedenih funkcija poziva, jednu za drugom.

Funkcije za uređivanje visina, bojanje i dekoriranje segmenata ovisiti će o tome koje se klase koje implementiraju apstraktne klase OW\_Sculptor, OW\_Painter, OW\_Decorator nalaze na istom GameObject objektu kao i instanca OW\_ChunkEditor. Same klase OW\_Sculptor, OW\_Painter, OW\_Decorator samo su apstraktne klase koje se koriste za grupiranje objekata koji su instance klasa koje ih nasljeđuju; potrebno je implementirati novu klasu koja nasljeđuje jednu od njih kako bi se mogla koristiti pri uređivanju terena.

Svaka apstraktna klasa od navedene tri ima slično-imenu javnu funkciju koja se poziva unutar OW\_ChunkEditor instance:

* OW\_Sculptor.Sculpt(OW\_Chunk chunk, int seed)
* OW\_Painter.Paint(OW\_Chunk chunk, int seed)
* OW\_Decorator.Decorate(OW\_Chunk chunk, int seed)

Osim toga, apstraktne klase imaju i cjelobrojno polje Priority (prioritet), koje određuje kojim redoslijedom će se njihove funkcije uređivanja izvršavati. Što je niži broj, to je viši prioritet, te će se funkcija tog objekta prije izvršiti.

### OW\_PathPainter.cs

Ova varijacija OW\_Painter skripte zadužena je za iscrtavanje puteljaka koji su proceduralno generirani uz pomoć "seed-a". Sama ideja i tijek generacije je sljedeći:

* određivanje POI-a (points of interest);
* pronalaženje puteva između POI-a;
* bojanje tih puteva.

POI točke su točke koje se proceduralno generiraju, sami putevi i bojanje se može prilagoditi vlastitim potrebama unutar OpenWorlds UI-a na samoj skripti.

POI točke se generiraju za svaki segment koji je učitan, te nema preklapanja sa drugim segmentima. Generirani putevi protežu se samo preko POI-a vlastitog segmenta, odnosno nema puteva koji se stvaraju preko više segmenata.

Sama skripta je podijeljena na 6 cjelina:

* CLASS\_DATA,
* MAIN\_PAINT\_FUNCTION (poziv svih funkcija),
* POI\_HANDLING,
* PATH\_FINDING,
* RENDERING\_PATHS
* DEBUG\_VISUALIZATION

U POI\_HANDLING odvija se proceduralno generiranje POI-a. Metode u ovome dijelu su:

*private float SampleHeightFromMesh(Mesh mesh, Vector3 position)*

* metoda koja određuje (na temelju x i z pozicije) y vrijednost na mreži za potencijalan POI.

*private void EvaluateMinHeightThreshold(Mesh mesh)*

* metoda koja dinamički određuje minimalHeightThreshold sa funkcijom Math.Lerp()

*private List<Vector3> GeneratePOIs(OW\_Chunk chunk, int poiCount)*

* metoda koja preko Random.Range() dobiva x i z koordinate nekog kandidata za POI, tada se poziva funkcija SampleHeightFromMesh koja vraća odgovarajuću y koordinatu za POI kandidata.

U PATH\_FINDING odvija se traženje puteva. Putevi se sastoje od početnog i završnog POI-a, a između njih se traže točke koje bi mogle činiti put. Ovaj dio se sastoji od 4 metoda:

*private List<List<Vector3>> ConnectPOIs(List<Vector3> pois, OW\_Chunk chunk)*

* ovo je glavna metoda ovog dijela, ona prima popis POI-a u trenutnom chunku i prolazi kroz sve njihove kombinacije, određuje put od početnog do krajnjeg POI-a pomoću A\* algoritma

*private float Cost(Vector3 a, Vector3 b, OW\_Chunk chunk)*

* metoda primarno služi za određivanje cijene za A\* algoritam

*private List<Vector3> GetNeighbors(Vector3 node, OW\_Chunk chunk)*

* metoda prima točku unutar puta za koju se moraju odrediti susjedi, oni se radi optimizacijskih razloga traže na udaljenostima od 5 do 10 jedinica udaljenosti

*private List<Vector3> ReconstructPath(Dictionary<Vector3, Vector3> cameFrom, Vector3 current)*

* metoda na temelju stvorenog rječnika točke i njezinog roditelja rekonstruira put kojim se došlo od početnog POI-a do završnog

U RENDERING\_PATHS se odvija bojanje točaka koje su stvorene unutar puteva. Točke unutar puteva su međusobno udaljene pa nije samo cilj obojati samo tu točku nego se mora i područje oko njih obojati da bi se te praznine između dviju točaka puta popunile i da bismo dobili pravi put. Iz tih zahtjeva se onda implementiralo radijalno bojanje točaka puta. Za to se koriste početna i završna boja iz kojih se vuču gradijenti između tih boja da putevi izgledaju prirodniji i nisu jednobojni. Ovaj dio se sastoji od samo 1 metode:

*private void RenderPathWithColor(List<Vector3> path, OW\_Chunk chunk, Color pathColor, Color endColor)*

* metoda radi interpolaciju boje, određuje nasumičnu veličinu radiusa na temelju pathRadius i pathRadiusVariance
* metoda traži po listi vertices od chunk mesha sve točke koje su unutar radijusa bojanja te provodi bilinearno uzorkovanje da bi se ispravile nekonzistentnosti prilikom bojanja (pikseli se zaokružuju na cijele brojeve)

# Primjeri implementirane funkcionalnosti

U ovome poglavlju moguće je vidjeti nekoliko slika ekrana iz demonstracijskog projekta koje prikazuju implementiranu funkcionalnost alata PixFER i OpenWorlds.



Slike 3.1 i 3.2: Prikaz scene is demonstracijskog projekta iz dvije različite pozicije[[5]](#_Slike)



Slika 3.3: Prikaz otočne scene is demonstracijskog projekta[[5]](#_Slike)



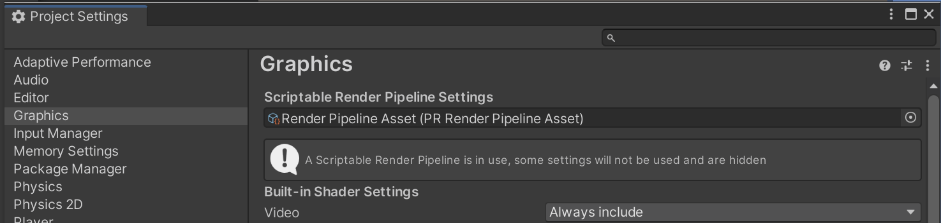
Slika 3.3: Prikaz šumske/planinske scene is demonstracijskog projekta (bliža perspektiva)[[5]](#_Slike)

# Upute za korištenje

## Korištenje PixFER alata u drugim Unity projektima

Kako bi se alat PixFER koristio u drugim Unity projektima potrebno je napraviti nekoliko koraka:

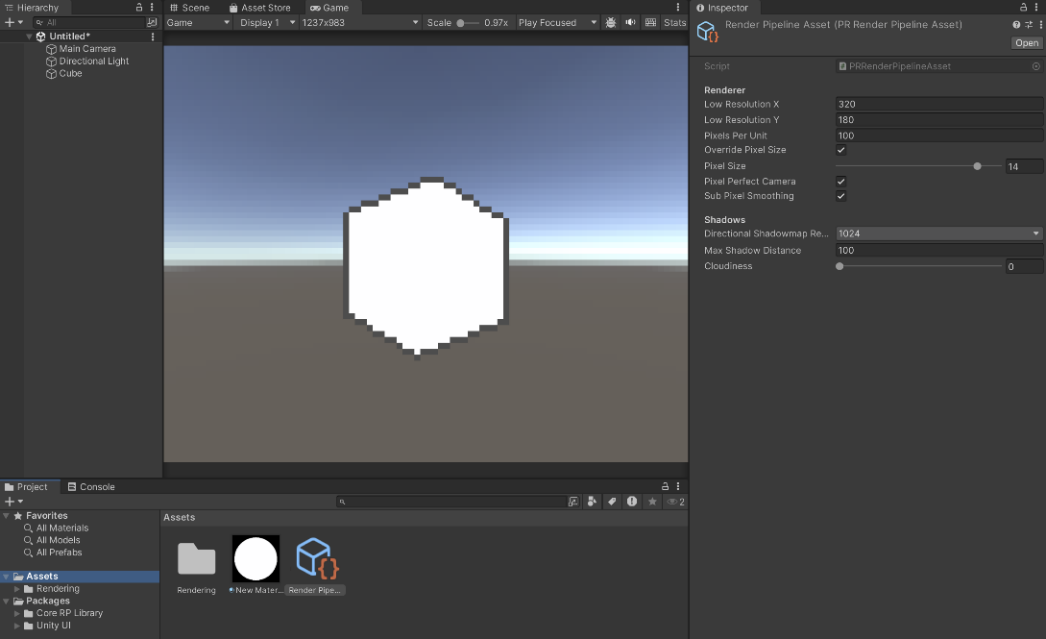
1. Instalirati “Core RP Library” paket dostupan u Unity Package Manager prozoru.
2. Uvesti paket “Rendering” koji sadrži sve glavne klase alata.
3. Napraviti objekt vrste “Rendering/PR Render Pipeline Asset”.
4. Objekt zatim postaviti kao atribut “Scriptable Render Pipeline Settings” u postavkama “Project Settings/Graphics”.



Slika 4.1: Prikaz postavljenog atributa u postavkama “Project Settings/Graphics”[[5]](#_Slike)

NAPOMENA: Nakon što je Render Pipeline ažuriran, objekti na sceni trebali bi postati nevidljivi. S time znamo da je svaki korak do sada uspješno napravljen.

1. Izraditi barem jedan novi materijal.
2. Na (jedan ili više) novih materijala potrebno je postaviti parametar sjenčara na jedan od onih pod kategorijom “Project R”.
3. Dodati novo-stvorene materijale na objekte u scenu kako bi postali ti objekti postali vidljivi.
4. Postaviti glavnu kameru na ortografski način prikazivanja i osnovna scena trebala bi biti spremna za rad.

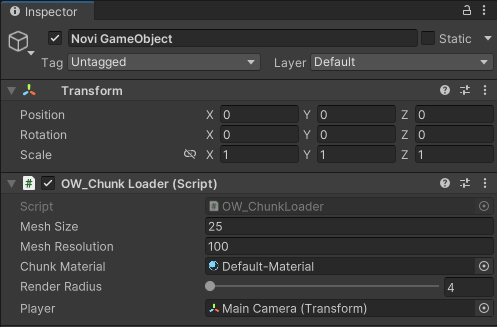


Slika 4.2: Primjer osnovne scene s prilagođenim SRP-om[[5]](#_Slike)

## Korištenje OpenWorlds alata u drugim Unity projektima

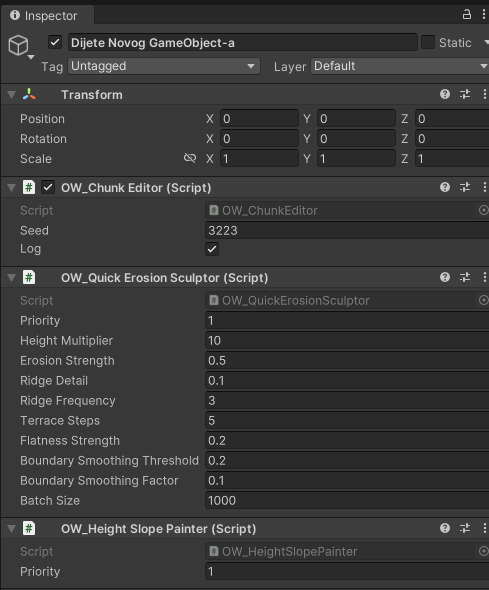
Kako bi se koristio OpenWorlds alat u drugim Unity projektima potrebno je napraviti sljedeće korake:

1. Uvesti paket “OpenWorlds” koji sadrži sve glavne klase alata.
2. Stvoriti novi, prazni GameObject.
3. Na novi GameObject staviti komponentu OW\_ChunkLoader.
4. Postaviti vrijednost ChunkMaterial na materijal iz Project Asset-a te postaviti vrijednost Player na objekt iz aktivne scene.

.

Slika 4.3: Primjer novog GameObject objekta s komponentom OW\_ChunkLoader[[5]](#_Slike)

1. Napraviti prazan GameObject objekt koji je dijete objekta iz prošlih koraka.
2. U “dijete” objekt staviti skriptu OW\_ChunkEditor i postaviti parametre Seed (nasumična generacija svijeta), te Log (u slučaju da želite aktivno ispisivati što se događa kada se segmenti stvaraju)
3. Stvoriti nove skripte koje nasljeđuju jednu od apstraktnih klasa OW\_Sculptor, OW\_Painter, OW\_Decorator te implementirati potrebne funkcije (moguće je koristiti i skripte koje dolaze s alatom).
4. Nove skripte postaviti kao komponente u “dijete” objekt.



Slika 4.4: Primjer “dijete” GameObject objekta s komponentama

OW\_ChunkEditor i OW\_QuickErosionSculptor.[[5]](#_Slike)

Prateći ove korake će OpenWorlds alat raditi u sceni u koju je dodan.

# Literatura

* Archer, Travis. "Procedurally generating terrain." *44th annual midwest instruction and computing symposium, Duluth*. 2011.
* Olsen, Jacob. "Realtime procedural terrain generation." (2004): 31.
* Valencia-Rosado, Luis Oswaldo, and Oleg Starostenko. "Methods for procedural terrain generation: a review." *Pattern Recognition: 11th Mexican Conference, MCPR 2019, Querétaro, Mexico, June 26–29, 2019, Proceedings 11*. Springer International Publishing, 2019.
* Cooper T. Scriptable Render Pipeline Overview. Unity Blog. URL: https://blog.unity.com/technology/srp-overview (date of access: 02.03.2024).
* CAUÁS, Daniel Cândido. "A modern approach for rendering 3D top-down games in pixel art." (2023).
* Inglis, Tiffany C., Daniel Vogel, and Craig S. Kaplan. "Rasterizing and antialiasing vector line art in the pixel art style." *proceedings of the symposium on non-photorealistic animation and rendering*. 2013.
* Cui, Xiao, and Hao Shi. "A\*-based pathfinding in modern computer games." *International Journal of Computer Science and Network Security* 11.1 (2011): 125-130.

# Slike

[1] – Slika preuzeta s web-stranice Grid-Paint (<https://grid-paint.com/images/details/5353898643554304>) i uređena u alatu MSPaint. Sve zasluge za sliku idu korisniku Nongly web-stranice Grid-Paint.

[2] – Slike napravljene pomoću generatora slikovnih UML dijagrama iz UML tekstualnog dijagrama na stranici planttext.com pod otvorenom licencijom.

[3] – Slike napravljene pomoću alata AstahUML pod AstahUML akademskom licencijom.

[4] – Slika napravljena pomoću alata Canva Whiteboard na web-stranici Canva.com bez korištenja sredstava pod licencijama.

[5] – Slike napravljene kao snimke zaslona unutar razvojnog okruženja Unity.